test-e.ro

CAIET ELEV

Lecția 23

Generarea termenilor unor expresii matematice. Șirul Fibonacci în Python. Aplicații

Două feluri de reguli prin care se naște un șir: termenul general, o expresie în care intră doar rangul, și relația de recurență, în care fiecare termen se sprijină pe cei dinaintea lui.

DISCIPLINA Informatică **CLASA** a IX-a · mate-info **LIMBAJ** Python 3

NUMELE ELEVULUI .....

CUM FOLOSEȘTI CAIETUL

Rezolvă exercițiile pe hârtie: urmărești execuția pas cu pas, completezi, asociezi și scrii singur soluțiile, apoi le verifici pe platformă. Ce ai voie să folosești: secvențial, if/elif/else, while, for cu range, %, //, liste (literal, indexare, len, for pe listă, .append), slicing, import math, min/max/sum, def/return. Fără dicționare, fără fișiere, fără clase. Fără atribuiri simultane: alunecarea se scrie pe rând, a = b apoi b = c. Fibonacci se inițializează cu a = 1, b = 0, valori care nu fac parte din șir. Fără diacritice în codul Python. Paritatea se scrie c % 2 == 0 sau c % 2 != 0.

CUPRINS

1 Ce știm deja despre construirea unui șir .....
2 Când formula spune totul despre un termen .....
3 Când un termen se sprijină pe cei dinaintea lui .....
4 Același șir, fără să ții minte tot .....
5 De la coadă la cap .....
6 Numărul ascuns în șir .....
7 Cod ghidat · cinci probleme .....

SECȚIUNEA 1 · REACTUALIZARE

1 Ce știm deja despre construirea unui *șir*

Știi să construiești o listă valoare cu valoare și să duci un rezultat de la un pas la următorul. Le reluăm pe amândouă acum, pentru că lecția de azi se sprijină exact pe ele.

O listă se pornește goală și crește pe măsură ce calculăm: `v = []`, apoi `v.append(valoare)` la fiecare pas. Numărul de elemente se află cu `len(v)`, iar elementul de pe poziția `i` se citește cu `v[i]`, pozițiile începând de la `0`.

Aceeași idee în Python (cu sintaxă reală):

```
v = []
for i in range(1, 6):
    v.append(i * i)
print(v)
```

A doua unealtă e variabila care **își păstrează valoarea de la un pas la altul**. La progresia aritmetică (L22), termenul următor se obține din cel curent: `t = t + r`. Variabila `t` nu se resetează la fiecare pas — ea „duce mai departe” rezultatul, iar bucla doar o împinge înainte.

> **DE REȚINUT**
>
> Într-o buclă, o variabilă declarată **înainte** de buclă supraviețuiește de la o iterație la alta. Una declarată **în interiorul** buclei se rescrie complet la fiecare pas. Toată lecția de azi stă pe prima variantă.

A EXERCIȚIUL A · GRILĂ

Recapitulare: liste și valori purtate

GRILĂ

Încercuiește răspunsul corect la fiecare întrebare.

1. Pentru `v = []` urmat de `v.append(4)` și `v.append(9)`, ce afișează `print(len(v))`?

   9 · 2 · 0 · 13

2. Pentru `v = [3, 6, 9, 12]`, ce afișează `print(v[1])`? 6 · 3 · 12 · 9

3. După secvența `t = 5` urmată de `for i in range(3): t = t + 2`, cât valorează `t`?

   2 · 15 · 11 · 7

4. Ce face metoda `append` asupra unei liste?

Adaugă elementul la începutul listei, împingând restul spre dreapta

Adaugă un element la sfârșitul listei, mărindu-i lungimea cu 1

Înlocuiește ultimul element al listei cu valoarea primită ca argument

Ordonează crescător elementele listei după fiecare adăugare

B

EXERCIȚIUL B · DRY-RUN

Cum crește o valoare purtată de la un pas la altul

DRY-RUN

Urmărește variabila t care „duce mai departe” rezultatul. Coloana i e dată; tu completezi valoarea lui t după ce s-a executat linia t = t + r. Ultimul rând arată starea în care bucla se oprește.

```
a = 5
r = 3
n = 5
t = a
for i in range(n - 1):
    t = t + r
print(t)
```

i	t după t = t + r
— (înainte de buclă)	5
0	
1	
2	
3	
4 → condiția i < 4 e falsă, bucla se oprește	17

HINT *Recitește linia t = t + r: valoarea nouă a lui t se calculează din valoarea lui de dinainte, nu din i. Pornește de la 5 și adună 3 la fiecare rând.*

SPRIJIN · SCRIE CODUL

Primii n multipli de 3

SCRIERE COD

Se citește un număr natural n (n ≥ 1). Afișează pe același rând, separate prin câte un spațiu, primii n multipli nenuli ai lui 3, construindu-i mai întâi într-o listă. Ex.: pentru n = 5 se afișează 3 6 9 12 15.

HINT *Ordinea contează: multiplii se afișează crescător, în ordinea în care au fost adăugați în listă.*

SOLUȚIA TA · PYTHON

SPRIJIN · SCRIE CODUL

Al n-lea termen al unei progresii

SCRIERE COD

Se citesc, de pe câte un rând, trei numere naturale: a (primul termen), r (rația) și n (n ≥ 1). Afișează al n-lea termen al progresiei aritmetice care începe cu a și crește din r în r. Ex.: pentru a = 5, r = 3, n = 4, termenii sunt 5, 8, 11, 14, deci se afișează 14.

HINT **Atenție.** *Primul termen este chiar a, deci de la el până la al n-lea mai sunt doar n − 1 pași de adunare, nu n.*

SOLUȚIA TA · PYTHON

SECȚIUNEA 2 · TERMENUL GENERAL AL UNUI ȘIR

2 Când formula spune totul despre un *termen*

Unele șiruri se descriu printr-o singură expresie matematică: pui în ea rangul termenului și primești valoarea. Nu ai nevoie de niciun termen dinainte.

Un șir poate fi dat printr-un **termen general**: o expresie în care intră doar rangul `i` al termenului. Ca să generezi primii n termeni, parcurgi rangurile de la 1 la n și evaluezi expresia pentru fiecare.

Exemplu: `t(i) = i · (i + 1) / 2`. Pentru `i = 1` obții 1, pentru `i = 2` obții 3, pentru `i = 3` obții 6, apoi 10, 15, 21… Sunt **numerele triunghiulare**: al i-lea dintre ele numără bilele așezate într-un triunghi cu i rânduri.

> **METODA TERMENULUI GENERAL**
>
> Pentru fiecare rang `i` de la 1 la n, calculezi `t = expresie(i)` și afișezi (sau adaugi în listă) rezultatul. Termenii sunt independenți: fiecare se poate calcula fără ceilalți, chiar și direct al 1000-lea.

Aceeași idee în Python (cu sintaxă reală):

```
n = int(input())
for i in range(1, n + 1):
    t = i * (i + 1) // 2
```

```
    print(t, end=" ")
```

Folosim `//` (împărțire întreagă), nu `/`, pentru că `t(i)` e mereu un număr întreg: dintre `i` și `i + 1`, unul e obligatoriu par, deci produsul se împarte exact la 2. Cu `/` am obține `1.0`, `3.0`, `6.0` — aceleași valori, dar scrise ca numere reale.

ATENȚIE

`range(1, n + 1)` merge de la 1 la n **inclusiv**, pentru că al doilea capăt nu se atinge niciodată. Dacă scrii `range(1, n)`, ultimul termen dispare fără niciun mesaj de eroare.

UNDE SE FOLOSEȘTE ÎN VIAȚA REALĂ

Formula de dobândă a unui depozit bancar îți dă suma după `i` luni fără să treci prin toate lunile dinainte. Într-un joc, punctajul necesar pentru nivelul `i` se calculează dintr-o expresie de `i` (de aceea nivelurile mari cer tot mai multe puncte). Iar un tabel de tarife („primul kilogram 10 lei, fiecare următor 4 lei”) e tot un termen general: `t(i) = 10 + 4 · (i − 1)`.

A

EXERCIȚIUL A · COMPLETARE

Cum se generează un șir dat prin formulă

COMPLETARE

Completează definiția folosind cuvintele din rezervă (două cuvinte rămân nefolosite).

`rangului` `condiție` `listă` `contorului` `expresie` `termen`

Un șir dat printr-o _____ matematică se generează evaluând-o pentru fiecare valoare a _____, de la 1 la n; fiecare rezultat obținut este un _____ al șirului și poate fi păstrat într-o _____, cu metoda append.

B

EXERCIȚIUL B · SCRIE CODUL

Numerele triunghiulare

SCRIERE COD

Se citește un număr natural n (n ≥ 1). Afișează pe același rând, separați prin câte un spațiu, primii n termeni ai șirului dat de expresia t(i) = i · (i + 1) / 2. Ex.: pentru n = 5 se afișează 1 3 6 10 15, pentru că 1·2/2 = 1, 2·3/2 = 3, 3·4/2 = 6, 4·5/2 = 10, 5·6/2 = 15.

HINT *Rezultatul este mereu un număr întreg, deci folosește // la împărțirea cu 2.*

SOLUȚIA TA · PYTHON

SECȚIUNEA 3 · ȘIRURI RECURENTE. FIBONACCI CU MEMORARE ÎN LISTĂ

3 Când un termen se sprijină pe cei dinaintea *lui*

La unele șiruri, formula nu conține rangul, ci termenii deja calculați. Cel mai cunoscut dintre ele are peste 800 de ani și apare în cochilii, flori și în design.

O **relație de recurență** spune cum se obține un termen nou din termenii calculați înainte. Ea vine mereu la pachet cu **termenii de start**, fără de care regula n-ar avea de unde porni.

> **ȘIRUL LUI FIBONACCI**
>
> F(1) = 1, F(2) = 1, iar de la al treilea termen încolo F(n) = F(n−1) + F(n−2): fiecare termen este suma celor doi dinaintea lui. Primii termeni: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144.

Șirul ajunge în Europa în 1202, prin cartea *Liber Abaci* a lui Leonardo Pisano, zis Fibonacci, dintr-o problemă despre înmulțirea iepurilor: numărul de perechi dintr-o lună e suma perechilor din ultimele două luni. Era însă cunoscut cu secole mai devreme matematicienilor indieni (Pingala, Virahanka, Hemachandra), care îl foloseau la numărarea tiparelor ritmice din poezie.

Prima implementare păstrează **toți** termenii într-o listă. Pornim lista cu cei doi termeni de start, apoi, pentru fiecare poziție nouă `i`, adăugăm suma celor de pe pozițiile `i - 1` și `i - 2`:

```
n = int(input())
f = [1, 1]
for i in range(2, n):
    f.append(f[i - 1] + f[i - 2])
```

```
for x in f:
    print(x, end=" ")
```

Poziția `i` din listă corespunde termenului F(i+1), pentru că indicii încep de la 0: `f[0]` e primul termen, `f[1]` al doilea, `f[6]` al șaptelea. Relația `f[i] = f[i-1] + f[i-2]` e traducerea directă, literă cu literă, a formulei F(n) = F(n−1) + F(n−2).

ATENȚIE

Lista pornește deja cu **doi** termeni, deci varianta aceasta are sens numai pentru n ≥ 2 — de aceea enunțurile o garantează. Bucla începe de la `i = 2` pentru că primele două poziții sunt deja ocupate; dacă ar începe de la 0, `f[i - 2]` ar deveni `f[-2]`, iar Python ar citi liniștit de la coada listei și ar produce un șir greșit, fără niciun mesaj de eroare.

UNDE SE FOLOSEȘTE ÎN VIAȚA REALĂ

Numărul de petale al florilor comune este de foarte multe ori un număr Fibonacci: 5 la piciorul-cocoșului, 13 sau 21 la margarete, 3 petale (și 3 sepale) la crin. Nu este însă o lege — macul și mușcatul au 4 petale. În schimb, semințele de floarea-soarelui se așază, fără excepții cunoscute, în spirale al căror număr e tot Fibonacci. În informatică, structurile de date „Fibonacci heap” îmbunătățesc, în teorie, timpul algoritmului de drum minim și își iau numele de la acest șir; în practică, aplicațiile de navigație folosesc alte structuri, mai simple. Unii investitori trasează „niveluri Fibonacci” pe graficele de preț — o practică populară, dar fără fundament științific demonstrat. E un exemplu bun pentru o întrebare pe care merită să o pui oricărei aplicații a matematicii: *există dovezi sau doar obicei?*

A

EXERCIȚIUL A · GRILĂ

Regula de recurență

GRILĂ

Încercuiește răspunsul corect la fiecare întrebare.

1. Care este al șaptelea termen al șirului lui Fibonacci, dacă F(1) = 1 și F(2) = 1? (13) (8) (21) (11)

2. Ce înseamnă că un șir este definit printr-o relație de recurență?

   - `Fiecare termen se calculează dintr-o expresie care conține doar rangul lui`
   - `Termenii se repetă periodic, de la un moment dat încolo, în aceeași ordine`
   - `Fiecare termen nou se calculează din termenii dinaintea lui`
   - `Toți termenii se obțin înmulțind termenul dinainte cu o valoare fixă`

3. Pentru `f = [1, 1, 2, 3, 5]`, ce valoare are `f[4] + f[3]`? (5) (9) (8) (7)

4. De câți termeni de start are nevoie regula F(n) = F(n−1) + F(n−2) ca să poată porni?

Unul singur, pentru că restul se deduc din el prin adunare

Niciunul, pentru că formula se calculează singură de la zero

Doi — primii doi termeni, amândoi egali cu 1

Trei, câți termeni apar scriși în formulă

B

EXERCIȚIUL B · DRY-RUN

Cum crește lista, poziție cu poziție

DRY-RUN

Urmărește cum se umple lista f pentru n = 7. Coloanele i și f[i-2] sunt date; tu completezi valoarea lui f[i-1] și termenul nou care se adaugă în listă. Ultimul rând arată starea în care bucla se oprește.

```
n = 7
f = [1, 1]
for i in range(2, n):
    f.append(f[i - 1] + f[i - 2])
print(f)
```

i	f[i-2]	f[i-1]	termen nou f[i-1] + f[i-2]
2	1		
3	1		
4	2		
5	3		
6	5		
7 → condiția i < 7 e falsă, bucla se oprește	—	—	—

HINT *Verifică ce ai deja în listă înainte de fiecare pas. La i = 4, lista e [1, 1, 2, 3], deci f[3] = 3 și f[2] = 2, iar termenul nou e 5.*

C

EXERCIȚIUL C · SCRIE CODUL

Primii n termeni Fibonacci, cu memorare în listă

SCRIERE COD

Se citește un număr natural n (n ≥ 2). Afișează pe același rând, separați prin câte un spațiu, primii n termeni ai șirului lui Fibonacci, construindu-i într-o listă. Ex.: pentru n = 5 se afișează 1 1 2 3 5.

HINT **Atenție.** *Lista pornește cu doi termeni deja scriși, [1, 1], deci bucla care adaugă termeni noi începe de la poziția 2.*

SOLUȚIA TA · PYTHON

4 SECȚIUNEA 4 · FIBONACCI CU DOUĂ VARIABILE

Același șir, fără să ții minte *tot*

Ca să afli termenul următor n-ai nevoie de tot șirul — îți ajung ultimele două valori. Le ținem în două variabile care „alunecă” la fiecare pas.

Regula F(n) = F(n−1) + F(n−2) cere doar **doi** termeni în orice moment. Așa că, în loc de o listă întreagă, ținem trei variabile: `a` și `b` sunt purtătorii, iar `c` e termenul nou.

METODA CU DOUĂ VARIABILE

Pornim cu `a = 1` și `b = 0`, valori care **nu sunt termeni ai șirului la pozițiile pe care le generăm** — ele există doar pentru ca prima sumă `a + b` să iasă chiar primul termen. La fiecare pas calculăm `c = a + b`, îl afișăm, apoi alunecăm: `a` primește vechiul `b`, iar `b` primește pe `c`. Memoria folosită rămâne aceeași, indiferent câți termeni generăm.

Aceeași idee în Python (cu sintaxă reală):

```
n = int(input())
a = 1
b = 0
for i in range(n):
    c = a + b
    print(c, end=" ")
    a = b
    b = c
```

De ce tocmai `a = 1, b = 0`? Pentru că atunci primul `c = 1 + 0 = 1` este chiar primul termen, iar al doilea `c = 0 + 1 = 1` este al doilea. Cu `a = 1, b = 1` primul `c` ar fi `2` — șirul ar începe cu `2 3 5 8` și am pierde cei doi de 1 de la început.

ATENȚIE

Ordinea celor două alunecări nu e negociabilă: întâi `a = b`, apoi `b = c`. Dacă le inversezi, `b` devine `c` înainte ca `a` să apuce să ia vechea valoare a lui `b`, iar șirul iese greșit de la al treilea termen încolo.

LISTĂ SAU DOUĂ VARIABILE?

Cu listă păstrezi toți termenii și poți reveni oricând la ei (de exemplu ca să-i afișezi invers), dar memoria crește odată cu n. Cu două variabile memoria rămâne constantă, însă termenii vechi se pierd definitiv. Alege după ce ai nevoie *după* generare.

Curiozitate. În Python, cele două alunecări se pot scrie și pe un singur rând: `a, b = b, c`. Este atribuirea simultană, permisă începând cu lecția 20. La noi rămân două linii separate, ca să se vadă limpede ordinea pașilor.

UNDE SE FOLOSEȘTE ÎN VIAȚA REALĂ

Un senzor care măsoară temperatura la fiecare minut, timp de un an, nu poate ține toate valorile în memoria unui microcontroler — păstrează doar ultimele câteva și calculează din ele. Un ceas care afișează media pașilor pe ultimele două zile lucrează la fel. Iar în jocuri, „ultimele două poziții” ale unui personaj sunt tot ce trebuie ca să-i calculezi direcția, fără să reții tot traseul.

A

EXERCIȚIUL A · ORDONARE

Așază algoritmul în ordine

ORDONARE

Pașii de mai jos generează, în pseudocod, toți termenii Fibonacci mai mici sau egali cu n. Sunt amestecați — numerotează-i de la 1 la 8, în ordinea corectă, astfel încât algoritmul să funcționeze.

- [ ] cât timp a + b ≤ n execută
- [ ] sfârșit cât timp
- [ ] b ← c
- [ ] a ← b
- [ ] a ← 1; b ← 0
- [ ] scrie c
- [ ] citește n
- [ ] c ← a + b

B

EXERCIȚIUL B · SCRIE CODUL

Al n-lea termen Fibonacci, fără listă

SCRIERE COD

Se citește un număr natural n (n ≥ 1). Afișează al n-lea termen al șirului lui Fibonacci, folosind doar variabilele a, b și c, fără să construiești o listă. Ex.: pentru n = 7, termenii sunt 1, 1, 2, 3, 5, 8, 13, deci se afișează 13.

HINT *Nu ai nevoie să afișezi termenii intermediari — doar ultimul c calculat.*

SOLUȚIA TA · PYTHON

PERFORMANȚĂ · SCRIE CODUL

Câți termeni încap sub o limită

SCRIERE COD

Se citește un număr natural x (x ≥ 1). Afișează câți termeni ai șirului lui Fibonacci sunt mai mici sau egali cu x. Ex.: pentru x = 10, termenii care nu depășesc 10 sunt 1, 1, 2, 3, 5, 8 — deci se afișează 6.

HINT *Aici nu mai știi dinainte câți pași faci: numărul de termeni e chiar răspunsul căutat, deci bucla for se înlocuiește cu un while a cărui condiție privește valoarea termenului următor, nu rangul lui.*

HINT **Atenție.** *Cei doi termeni egali cu 1 de la începutul șirului se numără amândoi: pentru x = 1 răspunsul este 2, nu 1.*

SOLUȚIA TA · PYTHON

SECȚIUNEA 5 · AFIȘAREA ȘIRULUI ÎN ORDINE DESCRESCĂTOARE

5 De la coadă la *cap*

Odată ce ai șirul într-o listă, îl poți citi și invers. Sunt două metode, la fel de corecte, dar cu costuri diferite — le comparăm pe amândouă.

METODA 1 — INDICE DESCRESCĂTOR

Pornești indicele de la ultima poziție și cobori până la prima: `for i in range(len(f) - 1, -1, -1)`. Al treilea argument al lui `range` este **pasul**, iar un pas negativ înseamnă „numără înapoi”. Nu se creează nicio listă nouă — parcurgi aceleași elemente, în altă ordine.

```
1  f = [1, 1, 2, 3, 5]
2  for i in range(len(f) - 1, -1, -1):
3      print(f[i], end=" ")
```

ATENȚIE

Capătul din dreapta al lui `range` **nu se atinge niciodată**. De aceea scriem `-1`, nu `0`: cu `range(4, 0, -1)` ultimul indice vizitat ar fi 1, iar primul element al listei s-ar pierde din afișare.

METODA 2 — FELIEREA F[::-1]

Scrisă între paranteze drepte, felierea (*slicing*) cu pas `-1` construiește o **listă nouă**, cu aceleași valori așezate invers. Codul e scurt și limpede, dar ocupă în memorie încă o listă de aceeași mărime.

```
1  f = [1, 1, 2, 3, 5]
2  for x in f[::-1]:
3      print(x, end=" ")
```

Curiozitate. Există și `f.reverse()`, care întoarce lista **pe loc**: nu creează una nouă, dar modifică ireversibil lista existentă — după apel, ordinea inițială nu se mai poate recupera. De aceea îl folosim doar când chiar vrem să pierdem ordinea originală.

CÂT NE COSTĂ FIECARE METODĂ?

Ambele metode de mai sus parcurg lista o singură dată, deci fac **O(n)** pași — ordinele de complexitate le-ai întâlnit la lecția 18. Diferă la *memorie*: parcurgerea cu indice nu alocă nimic în plus, adică **O(1)** memorie suplimentară, în timp ce felierea `f[::-1]` construiește o listă nouă, de aceeași mărime — încă **O(n)**.

Există însă și o a treia cale, care *generează* termenii direct în ordine descrescătoare, fără nicio listă: pornind de la F(n) și F(n−1), fiecare termen anterior se obține prin scădere, F(k−1) = F(k+1) − F(k). Este tot **O(n)** ca timp, dar **O(1)** ca memorie. Iar dacă, pentru fiecare dintre cei n termeni afișați, ai regenera șirul de la 1 până la el, ai face **O(n²)** pași: la n = 1000, prima variantă face aproximativ o mie de pași, a doua un milion. Aceeași problemă, aceleași rezultate, de o mie de ori mai mult lucru.

UNDE SE FOLOSEȘTE ÎN VIAȚA REALĂ

Într-o aplicație de mesaje, conversația se salvează în ordinea în care au sosit mesajele, dar se afișează cu cel mai nou sus — adică invers. Un clasament sportiv se construiește crescător după timp și se citește descrescător după punctaj. Iar butonul „Undo” dintr-un editor parcurge lista acțiunilor exact de la coadă la cap.

A

EXERCIȚIUL A · ASOCIERE

Ce produce fiecare expresie

ASOCIERE

Pentru lista `f = [1, 1, 2, 3, 5]`, fiecărei expresii din stânga îi corespunde un singur rezultat. Scrie în căsuță litera rezultatului pe care îl produce expresia. Fiecare literă se folosește o singură dată.

☐	range(4, -1, -1)
☐	range(5, 0, -1)
☐	f[::-1]
☐	f[len(f) - 1]

A `valoarea 5` B `lista [5, 3, 2, 1, 1]` C `valorile 4 3 2 1 0` D `valorile 5 4 3 2 1`

B

EXERCIȚIUL B · COMPLETARE

Regula parcurgerii inverse

COMPLETARE

Completează regula folosind fragmentele din rezervă (două fragmente rămân nefolosite).

`f[0]` `f[::-1]` `-1` `len(f)` `-1` `len(f) - 1`

Pentru a parcurge o listă de la coadă la cap, indicele pornește de la _____, coboară până la primul element și primește pasul _____; capătul din dreapta scris în range este _____, pentru că el nu se atinge niciodată. Aceeași inversare, dar într-o listă nouă, se obține cu felierea _____.

C

EXERCIȚIUL C · SCRIE CODUL

Fibonacci în ordine descrescătoare

SCRIERE COD

Se citește un număr natural n (n ≥ 2). Construiește într-o listă primii n termeni ai șirului lui Fibonacci, apoi afișează-i pe același rând, separați prin câte un spațiu, în ordine descrescătoare — de la al n-lea până la primul. Folosește parcurgerea cu indice descrescător. Ex.: pentru n = 5, lista este [1, 1, 2, 3, 5], deci se afișează 5 3 2 1 1.

HINT **Atenție.** *Cei doi termeni egali cu 1 apar amândoi la sfârșitul afișării — șirul inversat nu are voie să piardă niciun element.*

SOLUȚIA TA · PYTHON

★

EXCELENȚĂ · SCRIE CODUL

Fibonacci descrescător, fără listă

SCRIERE COD

Se citește un număr natural n (n ≥ 2). Afișează primii n termeni ai șirului lui Fibonacci în ordine descrescătoare, pe același rând, separați prin câte un spațiu, **fără să construiești nicio listă**. Ex.: pentru n = 5 se afișează 5 3 2 1 1.

HINT *Generează întâi, cu două variabile, până ajungi la F(n). Apoi coboară: din F(k+1) și F(k) obții termenul dinainte prin scădere, F(k−1) = F(k+1) − F(k).*

HINT De ce merită. *Soluția cu listă folosește O(n) memorie; aceasta folosește O(1), la același număr de pași. Compară-le și argumentează care e potrivită dacă n ar fi un milion.*

SOLUȚIA TA · PYTHON

6 SECȚIUNEA 6 · PROPORȚIA DE AUR

Numărul ascuns în *șir*

Împarte fiecare termen la cel dinaintea lui. Rapoartele nu se opresc la o valoare exactă, dar se strâng tot mai aproape de un număr celebru: φ ≈ 1,618.

Luăm rapoartele a doi termeni consecutivi din șirul lui Fibonacci: 1/1 = 1, apoi 2/1 = 2, 3/2 = 1,5, 5/3 ≈ 1,6667, 8/5 = 1,6, 13/8 = 1,625, 21/13 ≈ 1,6154, 34/21 ≈ 1,6190, 55/34 ≈ 1,6176. Valorile sar când peste, când sub o anumită valoare, dar de fiecare dată mai puțin — se apropie de ea din ce în ce mai strâns.

PROPORȚIA DE AUR

Valoarea către care tind rapoartele F(n+1)/F(n) este φ = (1 + √5) / 2 ≈ 1,6180339887. În Python o calculăm cu funcția `sqrt` din modulul `math`, învățată la lecția 21.

```
1  import math
```

```
fi = (1 + math.sqrt(5)) / 2
print(fi)
```

Cât de repede se apropie? Diferența față de φ este 0,0180 la raportul 8/5, 0,0070 la 13/8, 0,0026 la 21/13, **0,00101** la 34/21 — deci *încă peste* o miime — și abia 0,00039 la 55/34. Începând cu al nouălea raport, eroarea coboară sub o miime — șase termeni (5, 8, 13, 21, 34, 55) au fost de ajuns ca precizia să crească de peste patruzeci de ori.

ATENȚIE

Raportul se calculează cu `/` (împărțire obișnuită, care dă un număr real), nu cu `//`. Cu `//` toate rapoartele ar deveni `1` sau `2` și convergența ar dispărea complet.

UNDE SE FOLOSEȘTE ÎN VIAȚA REALĂ

Grila proporției de aur din aplicațiile de editare foto îți arată unde să așezi subiectul ca imaginea să pară echilibrată. În design de logo-uri și de interfețe, dreptunghiul de aur stabilește raportul dintre lățimea unei bare laterale și restul ecranului. Iar în natură, unghiul dintre două frunze consecutive pe tulpină (aproximativ 137,5°, „unghiul de aur”) derivă din φ și este unghiul care împiedică cel mai bine suprapunerea frunzelor, lăsându-i fiecăreia acces la lumină.

A

EXERCIȚIUL A · GRILĂ

Rapoarte și proporția de aur

GRILĂ

Încercuiește răspunsul corect la fiecare întrebare.

1. Către ce valoare se apropie raportul dintre doi termeni consecutivi ai șirului lui Fibonacci?

   `Aproximativ 1,618 — proporția de aur, notată cu ϕ`

   `Aproximativ 3,142 — raportul dintre circumferință și diametru`

   `Aproximativ 2,718 — baza logaritmului natural, notată cu e`

   `Aproximativ 1,414 — rădăcina pătrată a lui 2`

2. Cât este raportul 13/8, adică F(7)/F(6)? `1,6` `1,615` `1,667` `1,625`

3. Care este formula exactă a proporției de aur?

`(1 + √5) / 2` `√5 / 2` `(1 + √2) / 2` `(1 + √5) / 5`

B

EXERCIȚIUL B · SCRIE CODUL

De la ce rang ne apropiem de φ

SCRIERE COD

Se citește un număr natural n (n ≥ 3). Afișează cel mai mic rang k, cu 1 ≤ k ≤ n, pentru care raportul F(k+1)/F(k) diferă de φ = (1 + √5)/2 cu mai puțin de 0,001. Dacă în primele n ranguri nu există un astfel de k, afișează 0. Ex.: pentru n = 10: rapoartele sunt 1; 2; 1,5; 1,6667; 1,6; 1,625; 1,6154; 1,6190; 1,6176; 1,6182, iar diferențele față de φ scad de la 0,618 până la 0,00015. Prima diferență mai mică decât 0,001 apare la k = 9, deci se afișează 9.

HINT *Diferența se ia în valoare absolută, cu abs, pentru că rapoartele sunt când mai mari, când mai mici decât φ.*

HINT **Atenție.** *Ca să poți calcula raportul F(k+1)/F(k) pentru k = n, lista trebuie să aibă n + 1 termeni, nu n.*

SOLUȚIA TA · PYTHON

7 SECȚIUNEA 7 · COD GHIDAT

Scrie cod care *trece testele*

Cinci probleme. Scrie soluția pe hârtie, urmărește-o mental pe exemplul dat și abia apoi tastează-o pe platformă, unde e verificată automat pe 5 cazuri de test.

1

PROBLEMA 7.1 · COD GHIDAT

Suma primilor n termeni Fibonacci

SCRIERE COD

Se citește un număr natural n (n ≥ 1). Afișează suma primilor n termeni ai șirului lui Fibonacci. Ex.: pentru n = 5, termenii sunt 1, 1, 2, 3, 5, iar suma lor este 12.

HINT *Nu e nevoie de listă: aduni fiecare termen nou într-o variabilă acumulator, imediat ce l-ai calculat.*

HINT **Curiozitate.** *Suma primilor n termeni este întotdeauna cu 1 mai mică decât termenul de rang n + 2: pentru n = 10, suma e 143, iar F(12) = 144. Poți verifica singur cu programul tău.*

SOLUȚIA TA · PYTHON

2

PROBLEMA 7.2 · COD GHIDAT

Câți termeni pari

SCRIERE COD

Se citește un număr natural n (n ≥ 1). Afișează câți dintre primii n termeni ai șirului lui Fibonacci sunt numere pare. Ex.: pentru n = 10, termenii sunt 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, iar cei pari sunt 2, 8 și 34 — deci se afișează 3.

HINT *Paritatea se testează cu c % 2 == 0.*

HINT **Verifică singur.** *După ce rezolvi problema generând efectiv termenii, urmărește o observație: termenii pari par să apară din trei în trei. Compară numărul obținut de programul tău cu* `n // 3` *pentru mai multe valori ale lui n — se potrivesc de fiecare dată? Poți argumenta de ce? (Indiciu: urmărește tiparul par–impar–impar al parităților.)*

SOLUȚIA TA · PYTHON

3

PROBLEMA 7.3 · COD GHIDAT

Cel mai mare termen sub o limită

SCRIERE COD

Se citește un număr natural x (x ≥ 1). Afișează cel mai mare termen al șirului lui Fibonacci care este mai mic sau egal cu x. Ex.: pentru x = 100, termenii care nu depășesc 100 sunt 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, iar cel mai mare dintre ei este 89.

HINT **Atenție.** *Bucla se oprește când termenul următor ar depăși limita, deci răspunsul este ultimul termen calculat înainte de oprire — păstrează-l într-o variabilă separată.*

SOLUȚIA TA · PYTHON

PROBLEMA 7.4 · REPARĂ CODUL

4 Repară inițializarea

SCRIERE COD

Programul de mai jos ar trebui să afișeze primii n termeni ai șirului lui Fibonacci, pe același rând, separați prin câte un spațiu. Pentru n = 6 el afișează însă 2 3 5 8 13 21, în loc de 1 1 2 3 5 8. Găsește și repară greșeala, fără să schimbi structura buclei.

Programul stricat:

```
n = int(input())
a = 1
b = 1
for i in range(n):
    c = a + b
    print(c, end=" ")
    a = b
    b = c
```

Ex.: pentru n = 2, programul stricat afișează 2 3, iar cel reparat trebuie să afișeze 1 1.

HINT *Programul nu se blochează și nu dă niciun mesaj de eroare — produce liniștit un șir greșit. Sunt cele mai periculoase greșeli.*

HINT **Atenție.** *Valorile de start a și b nu sunt termeni ai șirului: ele există doar ca prima sumă a + b să iasă chiar primul termen.*

PROGRAMUL CORECTAT · PYTHON

PROBLEMA 7.5 · PROVOCARE

Șirul lui Tribonacci

SCRIERE COD

Șirul lui Tribonacci începe cu termenii 1, 1, 2, iar fiecare termen următor este suma celor trei dinaintea lui. Se citește un număr natural n (n ≥ 3). Afișează pe același rând, separați prin câte un spațiu, primii n termeni ai acestui șir. Ex.: pentru n = 6 se afișează 1 1 2 4 7 13, pentru că 1 + 1 + 2 = 4, apoi 1 + 2 + 4 = 7, apoi 2 + 4 + 7 = 13.

HINT *Se schimbă doar câți termeni intră în sumă și câți termeni de start sunt necesari; mecanismul rămâne exact cel de la Fibonacci.*

HINT **Atenție.** *Lista pornește cu trei termeni deja scriși, deci bucla care adaugă termeni noi începe de la poziția 3.*

SOLUȚIA TA · PYTHON